05

Controlando errores

Transcripción

[00:00] Vamos a hacer un pequeño overview comenzando desde la la clase 1 hasta el flujo que estamos llegando ahora con respecto, pues a la excepción qu hemos lanzado, que Java lanzó para nosotros. ¿Entonces qué tenemos? En un mundo ideal tenemos nuestro stack, que ejecuta main, después main método 1, main método 1, método 2, y una vez que ellos completan su flujo, automáticamente ellos son eliminados del stack y el stack queda vacío nuevamente.

[00:32] Esto en el mundo ideal. ¿Qué ha pasado aquí? ¿Qué es lo que pasó aquí? Aquí, voluntaria o involuntariamente, una bomba estalló. Entonces la bomba cayó aquí en método 2. Y como método 2 no sabía pues cómo trabajar con esa bomba, cómo desactivar esa bomba, la bomba cayó a método 1.

[00:58] Y a la vez, como método 1 tampoco sabía cómo desactivar esa bomba, la bomba llegó a main. Como main tampoco sabía cómo desactivar esa bomba, simplemente salió y la bomba explotó al final, y ninguno de esos métodos pud terminar su ejecución. ¿Ahora que sigue?

[01:17] Ahora tenemos que aprender cómo desactivar las bombas y Java nos da un mecanismo para avisarle, por ejemplo: "Java, yo creo que aquí puede haber una bomba". ¿Cómo? Si bien nosotros tenemos control de excepciones también con respecto a esos errores, en Java podemos también avisarle: "Java, yo creo que en este punto va a ocurrir un error".

[01:47] De repente puede ser que no, puede ser que yo no lo sepa y en efecto va a salir la excepción. Pero si yo creo que este código, por ejemplo, esa división entre 0, podría dar alguna bomba, algún error, yo le puedo avisar a Java. ¿Cómo? Java, tiene una palabra reservada también. Y esto sí es nuevo. Esa palabra se llama try. ¿Qué quiere decir try? Intenta. Yo le digo a Java: "Java, intenta esto, intenta aquello". Try. Try that.

[02:18] Entonces try tiene su propio bloque de ejecución, la estructura de try es esta. Le das un try y abres llaves, porque try tiene un bloque ejecución él solo, en el que él va a intentar hacer algo. Ahí le dices: "Java, intenta hacer esto". Y le vamos a decir que intente hacer esta división entre cero. Pero vemos que no compila. ¿Por qué?

[02:45] Porque try dice: "intenta esto, y si no da resultado, atrapa el error, atrapa la bomba". ¿Y cómo se llama esa otra palabra reservada? Catch. Atrapa. Y catch tiene un paréntesis, que es la excepción que él va a atrapar y también su propio cuerpo de ejecución, entonces nuevamente. Try es como decirle intenta, intenta esto. Catch, atrapa el error. Tenemos que recordarlo así, es una de las formas más fáciles de aprender cómo funciona el try catch.

[03:33] Un try no puede existir si no hay un catch. Es la primera regla, pregunta de examen de certificación. Try no puede existir sin catch. ¿Por qué? Porque él no puede intentar nada si es que en caso de error no hay nadie quien lo atrape, primera cosa. Vamos a pasar al catch. ¿Qué hay en el catch?

[03:55] Aquí yo le digo qué excepción él va a atrapar, entonces aquí yo le voy a decir que aquí atrape pues un ArithmeticException, y a este ArithmeticException yo le voy a llamar exception. ¿Qué les parece? Entonces yo aquí le digo que él atrape ArithmeticException, y como ya les dije que es un objeto entonces yo lo voy a declarar como una variable, exception.

[04:26] ¿Y cuál es la parte interesante aquí? Que en el catch, aquí, yo puedo trabajar con exception. ¿Por qué? Porque es un objeto. Y miren qué tiene

exception aquí al lado. Tiene método equals, tiene getLocalizedMessage, tiene getMessage. Entonces, si yo quiero saber qué sucedió, le voy a decir: "Okay, imprímeme el mensaje, imprime getMessage."

[04:53] Si es que tú atrapas esta excepcion, vamos a darle un poco de espacio aquí para que se vea más, perfecto. entonces aquí yo le voy a decir: "Si por alguna razón esto da este tipo de error ArithmeticException, entonces imprímeme el mensaje que te dio esa excepción, y también quiero saber por dónde pasó esa excepción".

[05:18] Entonces ya les dije que eso, ¿cómo se llamaba? Stack trace. Y para stacl trace yo ni siquiera necesito ponerlo dentro de un System.out.println, porque ya exception tiene un método llamado printStackTrace. ¿Qué hace este método printStackTrace? Imprime toda la pila por donde él pasó y nos da pues una vista un poco más amplia de qué es lo que sucedió. Básicamente esto que está aquí.

[05:50] Ahora, ¿cuál es la diferencia? Que como yo ya estoy atrapando esa excepción, él ya no debería explotar, la bomba ya no debería explotar, porque yo le estoy diciendo aquí cómo trabajar con esta bomba. Yo le estoy diciendo: "Aquí explota la bomba y vas a hacer esto con esta bomba, con esta bomba de aquí, ArithmeticException". Entonces vamos a ejecutar el código nuevamente.

[06:19] Guardamos y vemos, ahora sí, mira, inicio main, inicio método 1, inicio método 2, imprime 1, mensaje, by zero. ¿Por qué? Porque dio la excepción. Yo lo que le estoy diciendo aquí: "imprímeme el mensaje". ¿Él qué hizo después? Imprimió el stack trace. ¿Pero qué pasó? ¿Dónde está la maravilla? Que él aquí ya desactivó la bomba.

[06:45] Entonces el programa ?qué hace? Sigue su flujo. Entonces, aquí nuevamente llega aquí, 2. ¿Por qué? Porque él está siguiendo el for que está aquí iterando. Llega el for y él va a imprimir el número 2 y nuevamente yo

estoy lanzando aquí esta división entre cero, por lo tanto va a dar un error, y va a ser así sucesivamente hasta que esta condición ya no sea verdadera.

[07:14] Entonces él va a salir del for, va a terminar método 2, va a terminar método 1 y va a terminar main. Entonces, ¿qué es lo que está haciendo básicamente? Lo que está haciendo es, cuando la bomba llega a este punto, él la anula. Anulamos la bomba en método 2, por lo tanto la ejecución va a seguir tranquila, método 1 y main y main, porque la bomba ya fue anulada ahí.

[07:45] Vamos a hacer un debug rápido para ver cómo es que funciona esto, de una forma un poco más detallada. Nuevamente tenemos el punto aquí colocado, el stack comienza aquí, le vamos a dar step over, step in, llegamos acá, step over, step in, step over, step in para saber qué sucede y aquí era donde comenzaba o aquí entra el método incorrecto. Entonces lo que yo voy a hacer es salir de aquí. Perfecto.

[08:14] Con este que está aquí, que es step return, yo salgo de donde había entrado, otro tipo más. Y entramos aquí a iterar, entonces aquí yo le voy a dar step over, ya imprimió abajo, crea el número, hace la división. ¿Qué pasó? Intentó y atrapó. ArithmeticException, atrapó el error, porque él soltó esa excepción, pero soltó esa excepción y cayó donde yo quería que caiga.

[08:47] ¿Y aquí qué hago? Nuevamente step over, él imprime y el ciclo sigue. Es de esta forma que funciona el cath, y es así como podemos nosotros atrapar los errores y tener cierto control de los errores. Por ejemplo, si sabemos que este código de aquí puede dar ciertos errores, podemos también hacer esto de aquí. Vamos a comentar esto, vamos a darle un slash a cada uno para comentarlo.

[09:22] Y vamos a hacer otro tipo de error. Voy a crear un string test, y este string, como es un objeto, lo voy a declarar a nulo. ¿Y aquí qué voy a hacer? voy a darle un System.out.println, voy a darle test.toString(). Perfecto. Recuerden que él está referenciado a nulo, por lo tanto, él no apunta a nada y lo más

probable es que él imprima nulo aquí. Déjenme ver si él lanza un ArithmeticException o lanza algún otro tipo de error, vamos a probar aquí.

[10:09] Vemos que aquí él lanzó otro tipo de error y otra vez paró nuestra ejecución aquí. Entonces ¿qué sucedió aquí? No estamos preparados para un NullPointerException. ¿Por qué? Porque yo declaré aquí este string como nulo y aquí en test.toString, él no tenía nada que imprimir y dio una excepción, él dijo: "No, yo no sé qué imprimir, es una referencia, nula, vacía".

[10:36] Entonces lanza excepción. Catch preguntó: "¿Sabes cómo tratar con esa excepción?" No. Entonces lanza la excepción, lanza la bomba, porque no sé cómo desactivar una bomb a del tipo NullPointerException. Entonces, nuevamente, ¿qué podemos hacer en este caso?

[10:52] Java nos permite hacer otro catch, por ejemplo, pregunta de examen de certificación. Podemos hacer otro catch, y vemos que aquí me está sombreando en rojo por un error. ¿Por qué? Me dice: "Tú ya estás atrapando ese error aquí, a este nivel, entonces yo no puedo atrapar el mismo error dos veces. Pero sí puedo atrapar un error diferente".

[11:19] E incluso lo que yo puedo hacer aquí, para ver si es que atrapa el error correcto, es decirle aquí: "Atrapo Arithmetic", y voy aquí abajo ahora, y le voy a decir: "Atrapo NullPointer", para ver si es verdad que él está cayendo en el punto correcto. Guardamos nuevamente y le damos play. Mira lo que está imprimiendo: "Atrapo NullPointer", "Atrapo NullPointer".

[11:57] O sea, él está yendo al catch directo. Yo puedo incluso, si quiero ser un poco más avanzado, atrapar dos excepciones o más excepciones en el mismo catch. ¿Cómo? Llego aquí, copio esto y yo consigo aquí ya controlar las dos excepciones en una misma variable.

[12:21] Entonces, él aquí ya me da un error de compilación porque me dice: "Tú ya has atrapado ese error, entonces simplemente no puedes atrapar este error dos veces." ¿Qué hago? Borro este catch y aquí le voy a decir solamente:

"Atrapo Excepcion". Perfecto. Imprimo aquí. Guardo primero. Y dice "Atrapo Excepcion tipo NullPointer". "Atrapo Excepcion". Perfecto.

[12:50] Entonces, aquí tenemos nuestro stack trace, nuestra excepción y nuestro catch. Ya sabemos cómo tratar dos excepciones diferentes. Vamos a